ASSIGNMENT 6.3

Problem Statement:

- Explain Hive Architecture in Brief.
- Explain Hive Components in Brief.

Answer:

Hive Architecture and its components:

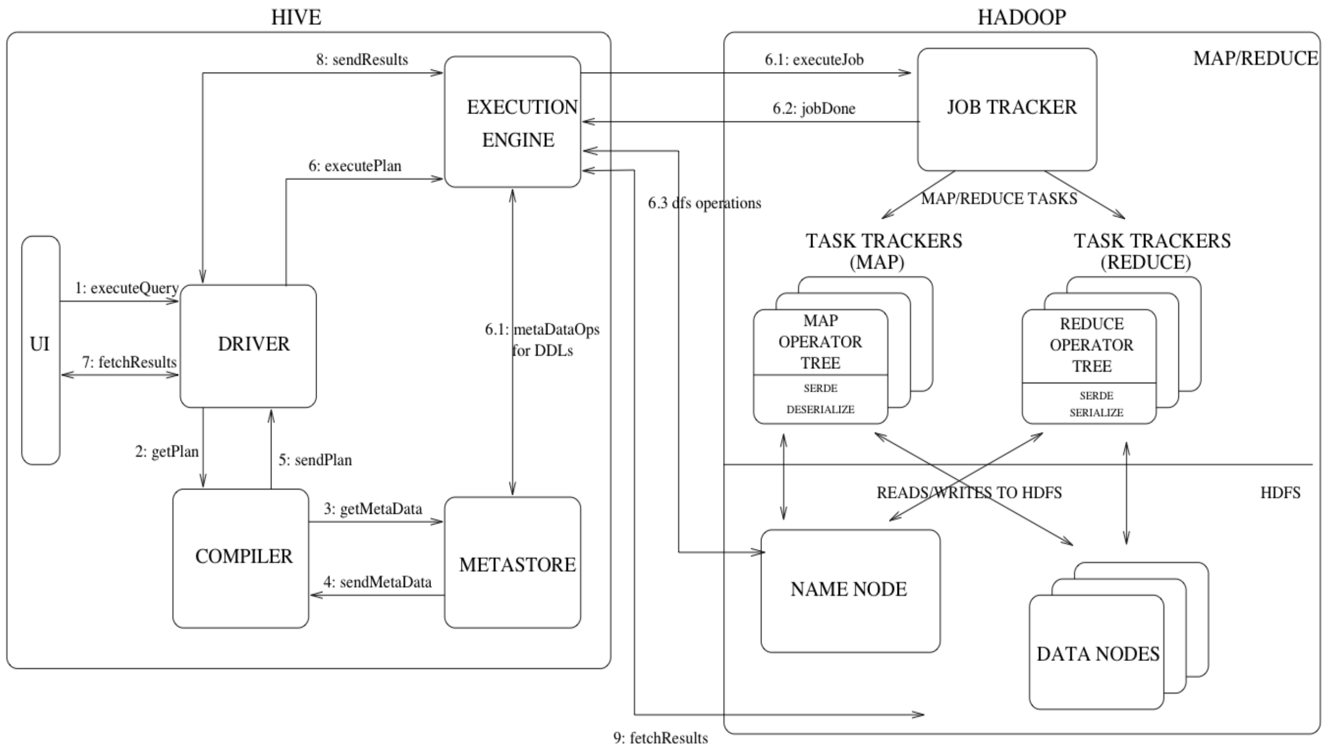

Figure: Hive architecture

The **main components of Hive system** are as follows:

- UI (Hive User Interface)
- Driver
- Compiler
- Meta store
- Execution Engine

User Interface: Users can submit their queries to Hive system via the user interface. The system has web based user interface and command line interface.

Driver: The driver component receives queries from users via user interface. It implements the process of handling sessions and it offers fetch and execute APIs built on JDBC/ODBC interfaces.

Compiler: This component parses a query, performs semantic analysis on various blocks and expressions of that query and finally produces an execution plan using the metadata related to tables and partitions from meta store.

Meta store: It stores information regarding structure of all the tables and partitions in the warehouse which contains columns and their type information, the serializers and deserializers required to read and write data and the related files on HDFS in which the data has been stored.

Execution Engine: As the name implies, it takes up the execution plan created by compiler and executes it. This plan will be in the form of DAG (Directed Acyclic Graph) stages. The execution engine handles the dependencies between these stages of execution plan and executes each of these stages on relevant components of the system.

The above figure indicates the flow of a query execution through Hive system.

Step 1: Initially, the user interface invokes execute interface to the Driver.

Step 2: The driver creates a session handle for the query submitted and transfers the query to compiler which generates execution plan.

Step 3 and 4: The compiler requests and obtains necessary metadata from the meta store.

Step 5: The metadata captured in previous steps will be used to type check the expressions in the given query and to prune partitions based on query predicates. The execution plan generated by the compiler is DAG of stages where each stage is a map/reduce job or a read write operation on HDFS.

Step 6, 6.1, 6.2 and 6.3: The execution engine submits the stages to appropriate components for execution. In each task of map or reduce operation the deserializer associated with the tables or intermediate outputs is used to read records from files on HDFS and these are passed through the corresponding operator tree.

Step 7, 8 and 9: Once the output gets generated, it is written to temporary HDFS file through the serializer. If the operation doesn't need a reducer, this task happens in the mapper. These temporary files are used to provide data to successive map/reduce stages of the plan. In case of DML operations the final temporary file will moved to table's location. This method is to make sure that dirty data is not being read. In case of queries, the execution engine reads the contents of temporary file directly from HDFS from the Drive as part of the fetch API call.